Report

이름: 김문겸

학번: 20220124

Lab: 2

Homework2-1: negate

```
int negate(int x) {
   return ~x + 1;
}
```

-x 값을 출력하는 것이므로, 2's complement 를 이용하여 return 해준다.

Homework2-2: isLess

```
                        int isLess(int x, int y) {
    int differ = ((~y+1)+x);
    int sign = ~(x^y);
    int plus = sign&differ;
    int must = (x&~y);

   return (plus|must)>>31&1;
}
```

Negate 를 참조하여 ((~y+1)+x), 즉 x-y 의 값을 구해줄 수 있다. 이때 >>31&1 을 통해서 x-y 의 부호 판별로 x 와 y 의 대소비교를 해줄 수 있지만, x=2147483647, y=-2147483648 일 때 x>y 이므로 0 이 나와야 하지만, 오버플로우 문제로 인해 1 이 출력된다. 이러한 케이스들을 고려하여, x=양수, y=음수와 같이 절대 x<y 가 성립할 수 없는 경우를 제외하는 방향으로 x<y 가 반드시 성립하는 case 들을 or 로 묶어 출력하면 구현이 가능할 것이라 생각했다. 이 코드의 경우 전체적으로는 x, y 의 부호가 같은 경우와 다른 경우를 나뉘어서 생각했다. differ 은 x-y 값을 의미하고, sign 은 x,y 의 부호의 동치 여부를 확인하는 변수이다. 같다면 1, 다르면 0 을 출력한다. Plus 는 sign&plus 의 값으로, sign 이 1 이라면(x,y 가 같다면), differ 가 양수라면 plus 가 1 이 되도록 하고, sign 이 0 이라면 후에 설명할 must 변수에서 판별하도록 0 을 출력한다. must 는 x 가 -, y 가 +인 경우에만 1 을 출력한다. 이 경우는 x, y 의 값이 어떠하든 반드시 x<y 인 경우이기 때문이다. 따라서 plus 변수에서 0 으로 출력되는 x, y 부호가 반대인 경우에는, x 가 음수, y 가 양수인 경우에만 1 이 출력되서 plus | must 값이 1 이 되고, >>31&1 연산을 통해 1 이 출력된다. 하지만 x 가 양수, y 가 음수인 경우에는 plus 와 must 둘 다 0 이 출력되어 최종적으로 0 이 출력된다.

Homework2-3: float_abs(unsigned uf){

```
unsigned float_abs(unsigned uf) {
    unsigned abs = uf&0x7FFFFFFF;
    unsigned e = abs&0x7F800000;
    unsigned f = abs&0x007FFFFF;
    if(e == 0x7F800000 && f != 0x00000000){
        return uf;
    }
    else{
        return abs;
    }
}
```

이 문제는 부동소수점의 절댓값을 구하는 문제이다. 부동소수점의 경우 가장 첫 번째 비트가 부호를 결정하고, 절댓값이 같은 양수와 음수가 첫 번째 비트를 제외한다면 모두 같음을 이용해서, 01111...111 = 0x7FFFFFFF 와 &연산하여 첫 번째 비트를 제외한 부분을 추출해서 출력하였다. 다만 문제에서 조건이 uf 가 NaN 일 경우 uf 그대로 출력하라라는 조건이 있었기 때문에, NaN 의 조건인 (1. Exp 부분이 모두 1 이어야 한다), (2. frag 부분이 0 이 아니어야 한다) 를 만족한다면 uf 를 출력, 그것이 아니라면 절댓값인 abs 를 출력한다.

Homework2-4: float_twice

```
unsigned float_twice(unsigned uf) {
    unsigned e = uf&0x7F800000;
    if(e == 0x7F800000){
        return uf;
    }
    else{
        if(e == 0x00000000){
            return ((uf&0x80000000) + (uf<<1));
        }
        else return (uf + 0x00800000);
    }
}
```

이 문제는 기존 uf 를 2 배한 값을 출력하는 문제이다. 2-3 번 문제와 마찬가지로 NaN 일 경우 argument 그대로 출력한다. NaN 이 아닐 경우, normalized 인지, denormalized 인지 판별한다. 이에 따라 2 배를 하는 방식이 달라지기 때문이다. Denormalized 의 경우 exp 부위가 모두 0 이고, frag 의 맨 앞자리가 1 부터 시작하기 때문에, uf 를 1 만큼 왼쪽으로 shift 하게 되면, frag 의 첫 번째 자리에 있던 1 이 exp 로 올라와서 exp 가 더 이상 모두 0 이 아니게 된다. 이렇게 되면 normalized 값이 되고, exp 가 0 에서 1 로 증가했기 때문에 2 배만큼 증가한 것이 된다. 이와 함께 부호를

상징하는 값인(uf&0x80000000)을 더해주면 최종적인 twice 값이 된다. 반면 normalized 의 경우, 기존 exp 에서 1 만큼 더해주기만 하면 된다. 이 것은 frag*(2^(exp))에서 frag*(2^(exp+1))이 된 것이므로 2 배 증가한 것이다.

Homework2-5: float_i2f

```
unsigned float_i2f(int x) {
    int sign = x&0x80000000;
    int f;
    int e;
    int shift = 31;
    int bias = 127;
    int up;

    if(x == 0){
        return 0;
    }
    if(sign == 0x80000000){
        x = ~x+1;
    }

    while(((x>>shift)&1) == 0){
        shift--;
    }
    x = x << (31 - shift);
    e = shift + bias;
    f = ((x>>8) & 0x007FFFFF);
    up = (x & 0x000000FF);

    if(up > 128){
        f++;
    }
    else if (up == 128){
        if(f&1 == 1){
            f++;
        }
    }

    if((f&0x00800000) == 0x00800000){
        f = 0;
```

```
        e++;
    }

    return sign + (e<<23) + f;
}
```

이 문제는 int 형식의 값을 float 형식, 즉 부동소수점 형식으로 바꾸는 것이다. 그래서 int 형식의 값에서 부동소수점의 부호, exp, frag 에 들어갈 int 의 MSB, 수를 나타내는 비트의 개수 등을 추출해야 한다. 가장 처음 부호를 얻어낼 수 있었다. 변수 sign 은 x 의 MSB 값으로, 부호의 정보가 된다. f 는 변환할 float 데이터의 frag, e 는 exp, shift 는 자릿수, up 은 부동소수점에서 반올림을 할지 말지에 대해 판별하기 위한 변수이다. 우선 예외처리를 해준다. X 가 0 일 경우, while 문에서 무한 루프에 빠지므로 x=0 이라면 바로 0 을 출력하도록 했다. 이후 음수인지 양수인지를 판별하고, 음수일 경우 절댓값을 추출하여 변환에 사용하도록 했다. 왜냐하면 부동소수점의 경우 절댓값이 같은 양수와 음수가 MSB 값만 다르고 나머지 부위는 다 똑같기 때문이다. 이후 해당 절댓값을 오른쪽으로 1 씩 shift 시키면서 0 이 될 때의 shift 횟수를 구한다. 이 것이 바로 비트 자리 수를 의미하기 때문이다. MSB 를 제외한 int 형의 비트 자릿 수는 31 개이기 때문에, 31 에서 shift 수를 뺀 만큼 x 를 왼쪽으로 shift 하게 되면 MSB 바로 다음 자리부터 차례대로 데이터 값이 작성된다. 그리고 x 를 8 번 오른쪽으로 shift 하여 frag 자리에 들어가도록 한다. 이후 해당 값을 반올림 할지 말지에 대한 여부를 결정한 후 그에 따라 frag 와 exp 값을 바꾼다, 이후 부호 값, exp 값, frag 값을 합쳐서 최종적인 float 변환값을 출력한다.

Homwork2-6:float_f2i

```
unsigned e = uf&0x7F800000;
    unsigned f = uf&0x007FFFFF;
    int sign = uf&0x80000000;
    sign = sign >> 31;
    if(e == 0x7F800000){ // infinity & NaN
        return 0x80000000u;
    }
    else if((e>>23)<0x0000007F){ // denormalized
        return 0x00000000;
    }
    else { // normalized
        unsigned real_e = (e>>23) - 0x0000007F;
        int temp;
        if(real_e >= 31){
            return 0x80000000u;
        }
        else if(real_e >= 23){
```

```
            temp = (f << (real_e - 23));
        }
        else{
            temp = (f >> (23 - real_e));
        }
        temp = temp + (1<<real_e);

        if((sign&1) == 0x00000001){
            return (-temp);
        }
        else return temp;


    }
```

이 문제는 5 번과 반대로 float 를 int 로 바꾸는 문제이다. 그래서 부동소수점을 3 등분하여 부호, exp, frag 로 나눈 다음, frag 값을 exp 만큼 shift 하여 int 형으로 변환하는 매커니즘을 생각했다. 가장 처음, 문제의 조건에 따라 NaN 과 infinity 값이 argument 로 들어오면 0x80000000u 를 출력하도록 했다. 그리고 float 의 경우 normalized 와 denormalized 의 형식이 다르므로, 두 조건을 나눈다. Denormalized 일 경우 0 보다 작은 매우 작은 값이기 때문에, int 형으로 나타낼 경우 0 을 출력하도록 한다. Normalized 일 경우, exp 를 통해 자릿수 값을 얻어, 자릿수가 frag 의 최대 자릿수인 23 보다 클 때와 작을 때를 나누어 shift 를 각각에 맞춰 취해 int 값을 얻을 수 있도록 한다. 마지막으로 부호 값과 합쳐주면, 최종적인 int 형 값이 된다.